Chapter

Field Programmable Reconfigurable Mesh (FPRM)

Esti Stein and Yosi Ben Asher

Abstract

Many application areas demand increasing amounts of processing capabilities. FPGAs have been widely used for improving this performance. FPRM (Field Programmable Reconfigurable Mesh) is a technique we propose to improve FPGA performance. A Reconfigurable Mesh (RM) consists of a grid of Processing Elements that use dynamic reconfigurations to create varying bus segments between them. The RM can thus perform computations such as Sorting or Counting in a constant number of steps. It has long been speculated that the RM's dynamic reconfigurations should replace the FPGA's static reconfigurations. We show that the RM is capable of not only speeding up specific computations such as sorting or summing, but also of speeding up the evaluation of Boolean circuits (BCs), which is the main purpose of the FPGA. Our proposed RM algorithm can evaluate BCs without causing size blowup. Furthermore, tri-state switching elements can be used instead of PEs in a grid.

Keywords: FPGA, reconfigurable mesh, DNF, Boolean circuits, tri-state

1. Introduction

FPGAs are integrated circuits that form a matrix of conigurable logic units (CLUs)[1] connected via programmable routing interconnects. By downloading different routing configurations to the FPGA, any circuit $C(x_0, \dots, x_{n-1})$ can be embedded and then executed/evaluated. After embedding the circuit's topology in the FPGA, the circuit is executed every time a new input is received. Due to the FPGA's routing interconnects and CLUs, this evaluation mode makes the FPGA relatively slow compared to ASICs.

Assuming the circuit $C(x_0, \dots, x_{n-1})$ that has been discussed previously, we wish to examine the possibility of speeding up the evaluation of C by using a dynamic mode of reconfiguration rather than the above-mentioned FPGA mode. Essentially, we devised an algorithm that evaluates $C(x_0, \dots, x_{n-1})$ faster than its depth [1] (the longest path from the root/output to any leaf/input) [2]. In a sequence of reconfiguration steps this algorithm: 1) Spans bus segments on different subsets of $\{x_0, \dots, x_{n-1}\}$ in parallel; 2) Uses a single broadcast in each of these bus segments, and computes in parallel the AND/OR/COUNTING-1 s(counting the # of '1's) of each

[1] See appendix A for all acronyms and abbreviations

[2] A preliminary version of the following algorithm and results was presented as a poster in [1]

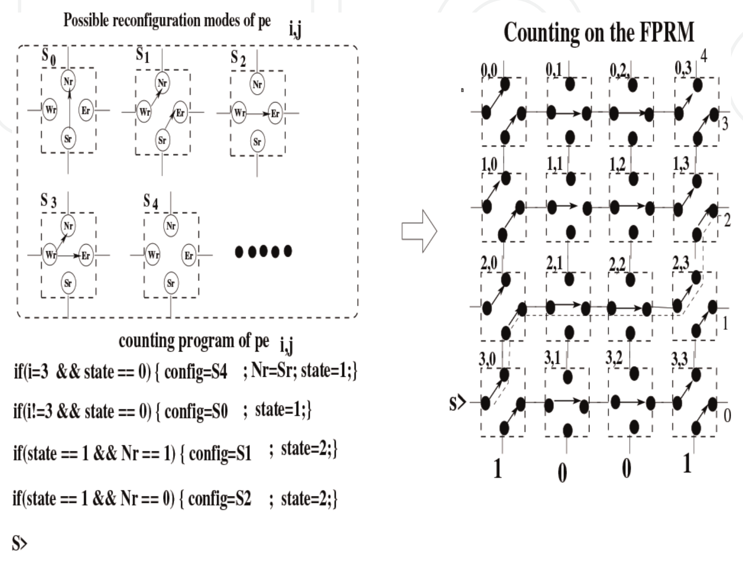
segment; 3) Computes C in a fixed small number of steps regardless of C's depth based on the above computations. The above algorithm uses a platform based on Reconfigurable Mesh (RM) [2], which is a 2D grid of Processing Elements (PEs) that uses dynamic bus reconfiguration to create varying bus segments for fast communication. Consequently, computations such as summation and sorting can be expedited.

Reconfigurable Mesh (RM) has been demonstrated to be able to perform parallel computations faster than the Parallel random access machine model (PRAM) [3], which is an abstract model for parallel computation. This includes $O(1)$ summing [4], $O(\log)$ integer summing [5], $O(1)$ multiplication [6], sorting [7], convex hall [8], graph algorithms [9, 10] and image processing [11]. Despite this potential power of the RM model, it has not yet been fully realized since the model assumes a signal can be transmitted along a bus/connected component in a single step regardless of the number of switches/ports. From this perspective, a variety of restricted RMs have been proposed. These include the RMBM [12], where only the structure of the RM's switch has been simplified but still busses with a linear number of switches are used. The SRGA [13, 14] proposed a mesh, where each row/column has a complete binary tree of reconfigurable switches, allowing to route messages between the leaves of this tree. [15] proposes a linear RM (LR-Mesh) bending cost, where the delay of a bus varies as a function of how many times it bends between rows and columns. It showed that for busses with a reasonable delay of at most $D = N^{\varepsilon}$ bends they can simulate algorithms for LR-Meshes in constant time. A bus of length $d(n) = n^{1/k}$ was suggested in [16], also showing that restricted RM algorithms can be directly coded in Verilog. This way of programming RM-algorithms overcomes most of the drawbacks of the C-like programming style proposed so far for RM-algorithms (e.g.,ARMlang [17]). However, they only addressed the problem of COUNTING-1 s. Our method of evaluating the circuit is partially based on the solution for COUNTING-1 s. [18] shows that integrating branching program with Boolean circuits is better than using each of them separately. Other realizations of the RM [19] were mainly to a small-size grid of Soft-CPUs and cannot be synthesized for large values of n.

A number of dynamic reconfiguration (DR) FPGAs have also been proposed, mainly for the purpose of speed acceleration. However, the main challenge was the reconfiguration delay. The use of DR is therefore rare [20]. There is also a method of addressing this problem, and it is commonly referred to as partial reconfiguration (PR) at runtime. PR can be implemented through external FPGA interfaces as well as special internal interfaces such as the ICAP on Xilinx devices [21]. Even so, PR is still primarily an auxiliary feature in modern commercial FPGAs rather than something with which the architecture is designed [22, 23]. Thus, PR design involves many details related to low-level architecture that require a high level of expertise. [24] proposed time-multiplexed DRFPGA, where registers are added to store computational states and partial results. Yet, only a few contexts are allowed because of area overhead. Memristors (RRAM), have also been applied as a programmable switch, as they are naturally more delay-efficient and lead to higher-performance FPGA architectures. However, [25, 26] only focus on the architectural repercussions of this technology. Very limited works investigate realistic RRAM-based circuit design constraints, while these have a strong impact on the final architectural performances. Fine-grain DR (FDR), described in [27], consists of homogeneous reconfigurable logic elements (LEs). It is possible to configure each LE as either a lookup table (LUT) or as an interconnect, or even as a combination of both. While this improves flexibility for allocating hardware resources between LUTs and interconnects, it still consumes a large amount of space. At first glance, this model seems to be close to what we have proposed, but one of the main difference lies in the algorithm for evaluating the Boolean Circuit $C(x_0, \ldots, x_{n-1})$ faster than its depth.

Our approach to the speed-up evaluation problem is to use the Reconfigurable Mesh (RM). We propose an infrastructure called FPRM (Field Programmable Reconfigurable Mesh) which is a sub-model of the RM model based on current CMOS technology and adapted to the proposed algorithm. The FPRM consists of two-dimensional grids of switches $pe_{i,j}$, with each switch connected to four neighbors $pe_{i-1,j}$,$pe_{i+1,j}$,$pe_{i,j-1}$,$pe_{i,j+1}$ via four links. It allows reconfiguration of its internal links in different reconfiguration modes S_0,S_1,S_2, … as depicted in **Figure 1** (upper left). Each $pe_{i,j}$ has four registers used to read/write to each of the four links: Nr to read/write to the link connecting $pe_{i,j}$ to $pe_{i-1,j}$ and $Sr/Wr/Er$ to read/write to $pe_{i+1,j}/pe_{i,j-1}/pe_{i,j+1}$ respectively. Each $pe_{i,j}$ executes a program based on its current state, its coordinates i,j and the values of Nr,Sr,Wr,Er. Upon execution, each $pe_{i,j}$ can change its reconfiguration mode, its state, and the content of its registers Nr,Sr,Wr,Er. **Figure 1** contains a four instructions program (bottom left side) for executing COUNTING-1 s of a four bits input. As depicted in **Figure 1** right side, the execution of this program creates a bus whose bendings corresponds to the $'1'$s input values. By examining the exit point (row number) of a signal sent through $S>$ we obtain the number of $1 - bits$ in the input.

The second step of the FPRM computation is shown in **Figure 2** computing the DNF (Disjunctive Normal Form): $((x_0 \wedge x_1 \wedge x_2) \vee (\overline{x}_0 \wedge \overline{x}_3) \vee (\overline{x}_1 \wedge \overline{x}_3) \vee (x_2 \wedge \overline{x}_3))$ where each and-term (minterm) is computed in a different row. As with COUNTING-1 s, we broadcast the input values along the columns in the first step. If $pe_{i,j}$ is associated with $\wedge x_i \wedge \ldots$ in an and-term (minterm) and the input $x_i == 1$ then $pe_{i,j}$ switches to a connect mode selecting S_2, alternatively on $(x_i == 0)$ it switches to a disconnect mode selecting S_4. The opposite is performed if $pe_{i,j}$ is associated with $\wedge \overline{x}_i \wedge \ldots$. A *true* signal is sent from $S>$ for every row, while each disconnected $pe_{i,j}$ broadcasts a *false* signal from its Er. The or-term of these and-terms is computed in another broadcast along the last column. Obviously, the FPRM can be used to execute the $O(1)$ RM algorithms such as summing of n numbers, multiplication [6, 28], sorting, convex hull [2], graph



```
if(i=3 && state == 0) { config=S4 ; Nr=Sr; state=1;}
if(i!=3 && state == 0) { config=S0 ; state=1;}
if(state == 1 && Nr == 1) { config=S1 ; state=2;}
if(state == 1 && Nr == 0) { config=S2 ; state=2;}
S>
```

Figure 1.
The FPRM switches and a program to compute COUNTING-1 s using a 4×4 *FPRM.*

algorithms [9] and image processing [11]). However, here we consider the problem of parallel evaluation of circuits with large depths for which no previous RM algorithm exists. Preliminary results demonstrate the FPRM feasibility and that it is likely to outperform FPGAs.

According to the proposed algorithm, boolean circuits $C(x_0, \ldots, x_{n-1})$ can be evaluated in a constant number of FPRM steps regardless of C's depth. During compilation, we calculate the minimized DNF formula dnf_y for every possible result of COUNTING-1 s, which is $y = \sum_0^{n-1} x_i$. An FPRM program is generated for each of the DNFs (dnf_y) using the algorithm described in **Figure 2**. Each of these DNFs (dnf_y) is compiled into an FPRM program working in a similar manner to the algorithm described in **Figure 2**. At run-time, after performing COUNTING-1 s of the input, the FPRM selects the DNF-program dnf_y for y and executes it. Thus, in a constant number of steps this algorithm computes $C(x_0, \ldots, x_{n-1})$, using dymamic reconfiguration (DR). By first applying COUNTING-1 s, we get that the size of the FPRM grid needed to execute each of the $dnf_{y=0,\ldots,n-1}$ is less or equal to the size of the original $C(x_0, \ldots, x_{n-1})$. This is expected since each dnf_y in $C(x_0, \ldots, x_{n-1})$ is restricted to the case where $y = \sum_0^{n-1} x_i$. Further, the and-terms of dnf_y are packed in a 2D-FPRM layout with multiple and-terms computed in a single row (unlike **Figure 2**, where each and-term is computed separately).

The rest of the chapter is organized as follows. The following section describes the use of COUNTING-1 s operation in order to reduce the formula size. The next step describes the problem of fitting as many and-terms as possible into an FPRM grid, which is one of the most challenging aspects of the technique. The results of the experiments will be presented next, followed by a summary of the conclusions.

2. Using the counting-1 s operation to reduce formula size

For every possible outcome of $y = \sum_0{}^* n - 1x_i$, the proposed algorithm starts by obtaining the minimized DNF formula, dnf_y. The input is divided into k segments containing *fracnk* bits, and the number of 1-bits is counted for each segment. For each of the $\left(\frac{n}{k}+1\right)^k$ possible COUNTING-1 s results $y_1, \ldots, y_k$ $\left(y_i = 0 \ldots \frac{n}{k}\right)$, we compute a minimized DNF $dnf_{y_1,\ldots,y_k}$ by:

1. Building a truth table $T_{y_1,\ldots,y_k}$ of $C(x_0, \ldots, x_{n-1})$ for all the binary numbers $x_0, \ldots, x_{n-1}$ with y_i '1's in the i segment.

2. Our initial DNF is formed by the nonzero entries of $T_{y_1,\ldots,y_k}$, which we simplify using the Logic Friday Espresso package [29].

3. Using reduced DNF, dnf_y, monochromatic rectangles $M \times V$ are created by searching for all minterms in M, and variables in V in such a way that any minterm in M contains all variables in V. We obtain a smaller version of dnf_y by replacing each variable in M by a new variable (which is the AND of all variables in M). A separate step needs to be performed in order to compute the value of the new variables corresponding to monochromatic rectangles.

Consider the truth table T of the address-function of $n = 6$ boolean variables given in **Figure 3**.

$$F(a,b,c,d,e,f) = \begin{cases} c & <a,b> = 0,0 \\ d & <a,b> = 0,1 \\ e & <a,b> = 1,0 \\ f & <a,b> = 1,1 \\ \end{cases}$$

T is arranged by COUNTING-1 s in $<a,b,c>$ $(y_1(a,b,c) \in \{0,1,2,3\})$, and COUNTING-1 s in $<d,e,f>$ $(y_2(d,e,f) \in \{0,1,2,3\})$. Since the address function has a very small formula to begin with

$$F(a,b,c,d,e,f) = ca'b' + da'b + eab' + fab$$

(where x' $is \neg x$), it is not expected that using COUNTING-1 s can significantly reduce the size of the remaining circuits $C^{y_1(a,b,c)=i, y_2(d,e,f)=j}(a,b,c,d,e,f)$. Indeed, the results in **Figure 3** shows that the minimal boolean formula for $C^{y_1(a,b,c)=2, y_2(d,e,f)=1}(a,b,c,d,e,f)$ is $a'bcde'f' + ab'cd'ef' + abc'd'e'f$ which is even larger than the original formula for the whole function $ca'b' + da'b + eab' + fab$. However, this happens only for four out of the sixteen possible cases of $y_1(a,b,c) = i, y_2(d,e,f) = j$. In all the remaining 12 cases the boolean formula has one or no variables.

Yet, COUNTING-1 s is very helpful for the multiplication function

$$F(a,b,c,d,e,f) = 1 \ \textit{iff} \ (a \cdot 2 + b) \cdot (c \cdot 2 + d)) \ mod \ 4 = (e \cdot 2 + f)$$

The results depicted in **Figure 4** shows that in all sixteen cases, the minimal boolean formula for $C^{y_1(a,b,c)=i, y_2(d,e,f)=j}(a,b,c,d,e,f)$ is very small.

Figure 5 depicts the largest dnf_y for $C = STCON(x_0, \ldots, x_{48})$ of a seven nodes directed graph where the input is 7×7 adjacency matrix of the graph. In this case we

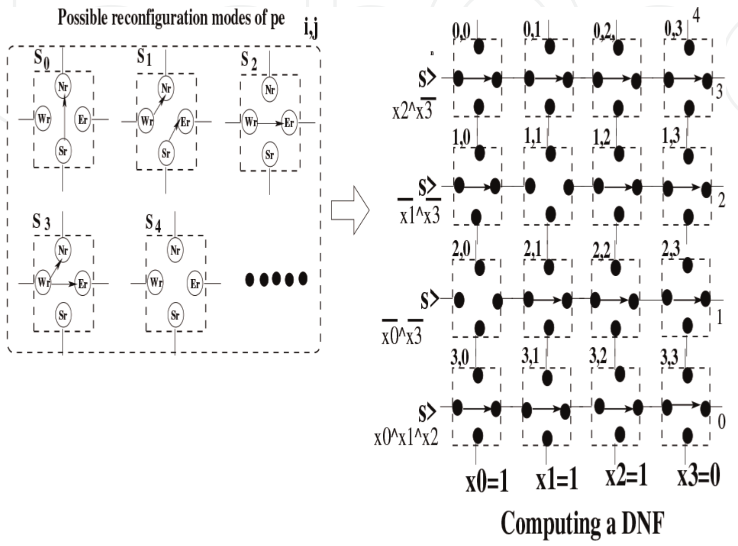
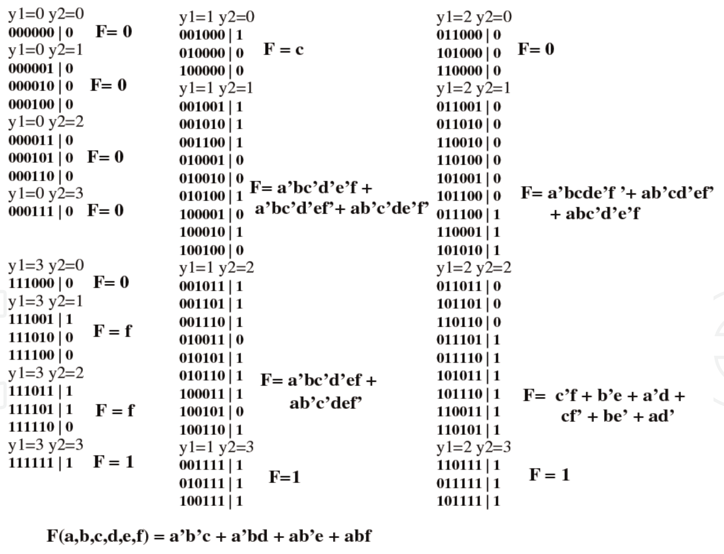


Figure 2.
Computing a DNF formula using a 4 × 4 FPRM.



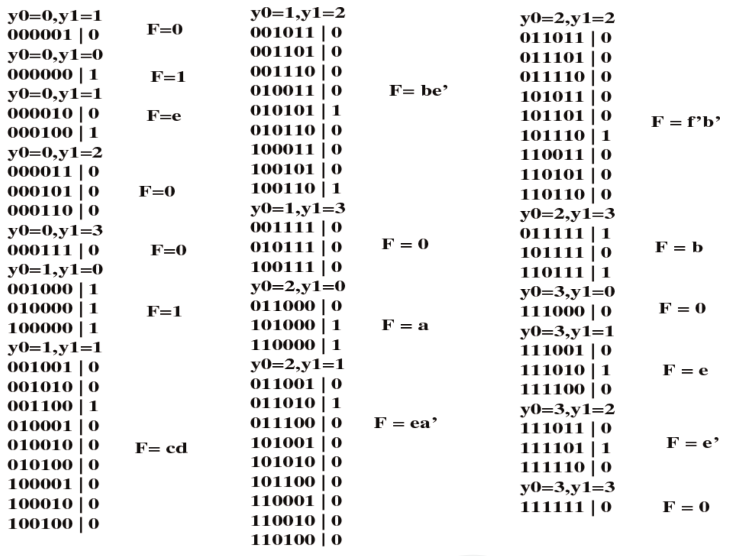
```
y1=0 y2=0
000000 | 0   F= 0
y1=0 y2=1
000001 | 0
000010 | 0   F= 0
000100 | 0
y1=0 y2=2
000011 | 0
000101 | 0   F= 0
000110 | 0
y1=0 y2=3
000111 | 0   F= 0

y1=3 y2=0
111000 | 0   F= 0
y1=3 y2=1
111001 | 1
111010 | 0   F = f
111100 | 0
y1=3 y2=2
111011 | 1
111101 | 1   F = f
111110 | 0
y1=3 y2=3
111111 | 1   F = 1
```

```
y1=1 y2=0
001000 | 1
010000 | 0   F = c
100000 | 0
y1=1 y2=1
001001 | 1
001010 | 1
001100 | 1
010001 | 0
010010 | 0   F= a'bc'd'e'f +
010100 | 1    a'bc'd'ef'+ ab'c'de'f'
100001 | 0
100010 | 1
100100 | 0
y1=1 y2=2
001011 | 1
001101 | 1
001110 | 1
010011 | 0
010101 | 1
010110 | 1   F= a'bc'd'ef +
100011 | 1       ab'c'def'
100101 | 0
100110 | 1
y1=1 y2=3
001111 | 1
010111 | 1   F=1
100111 | 1
```

```
y1=2 y2=0
011000 | 0
101000 | 0   F= 0
110000 | 0
y1=2 y2=1
011001 | 0
011010 | 0
110010 | 0
110100 | 0
101001 | 0
101100 | 0   F= a'bcde'f '+ ab'cd'ef'
011100 | 1        + abc'd'e'f
110001 | 1
101010 | 1
y1=2 y2=2
011011 | 0
101101 | 0
110110 | 0
011101 | 1
011110 | 1
101011 | 1
101110 | 1   F=  c'f + b'e + a'd +
110011 | 1        cf' + be' + ad'
110101 | 1
y1=2 y2=3
110111 | 1
011111 | 1   F = 1
101111 | 1
```

F(a,b,c,d,e,f) = a'b'c + a'bd + ab'e + abf

Figure 3.
Truth table of the address function arranged by COUNTING-1 s results.



```
y0=0,y1=1
000001 | 0   F=0
y0=0,y1=0
000000 | 1   F=1
y0=0,y1=1
000010 | 0   F=e
000100 | 1
y0=0,y1=2
000011 | 0
000101 | 0   F=0
000110 | 0
y0=0,y1=3
000111 | 0   F=0
y0=1,y1=0
001000 | 1
010000 | 1   F=1
100000 | 1
y0=1,y1=1
001001 | 0
001010 | 0
001100 | 1
010001 | 0
010010 | 0   F= cd
010100 | 0
100001 | 0
100010 | 0
100100 | 0
```

```
y0=1,y1=2
001011 | 0
001101 | 0
001110 | 0
010011 | 0   F= be'
010101 | 1
010110 | 0
100011 | 0
100101 | 0
100110 | 1
y0=1,y1=3
001111 | 0
010111 | 0   F = 0
100111 | 0
y0=2,y1=0
011000 | 0
101000 | 1   F = a
110000 | 1
y0=2,y1=1
011001 | 0
011010 | 1
011100 | 0   F = ea'
101001 | 0
101010 | 0
101100 | 0
110001 | 0
110010 | 0
110100 | 0
```

```
y0=2,y1=2
011011 | 0
011101 | 0
011110 | 0
101011 | 0
101101 | 0   F = f'b'
101110 | 1
110011 | 0
110101 | 0
110110 | 0
y0=2,y1=3
011111 | 1
101111 | 0   F = b
110111 | 1
y0=3,y1=0
111000 | 0   F = 0
y0=3,y1=1
111001 | 0
111010 | 1   F = e
111100 | 0
y0=3,y1=2
111011 | 0
111101 | 1   F = e'
111110 | 0
y0=3,y1=3
111111 | 0   F = 0
```

Figure 4.
Truth table of the mult function arranged by COUNTING-1 s results.

selected $k = \sqrt{49} = 7$, hence $y = <y_1, \dots ,y_7> \quad y_i = 0 \dots 7$. Out of all the COUNTING-1 s cases for $STCON(x_0, \ \dots, x_{48})$, **Figure 5** depicts the worst/largest dnf_y obtained. The dnf_y of **Figure 5** should be read as follows:

- *stcon e n* means STCON for graph of size n, while e is the number of entries in the adjacency matrix of the graph.
- *.i* e is the number of variables denoted by e (a variable for every entry in the adjacency matrix).
- *.p* m where m denotes the number of rows where the function is evaluated to TRUE.
- The rows are composed of e variables $a_0, \dots a_{e-1}$, where $-/1/0$ denotes $don't - care/a_i/a_{i'}$.

```
#stcon 49 7 .i 49 .o 1 .p 16
-------------1----------------------------------- 1
------11----------------------------------------- 1
------1--1-----00000----------------------------- 1
---------1----000000----------------------------- 1
------1---1-----------00000---------------------- 1
----------1----------000000---------------------- 1
--1----1------000000----------------------------- 1
---1---1-------------000000---------------------- 1
------1---1----00000--0-000---------------------- 1
------1--1-----00-00--00000---------------------- 1
--1-------1---000000--0-000---------------------- 1
----------1---000000-00-000---------------------- 1
---1-----1-----00-00-000000---------------------- 1
---------1----000-00-000000---------------------- 1
---1---1------000000-00-000---------------------- 1
--1----1------000-00-000000---------------------- 1
.e
```

Figure 5.
Resulting formula for worst COUNTING-1 s case of 7 × 7 STCON.

As shown in **Figure 5**, the dnf_y contains only 16 minterms (and-terms), each containing 10 variables (or negation of variables) omitting some $all\ don't - care$ columns at the end of each row. To compare, we obtained the full circuit for $STCON(x_0, \ldots, x_{48})$ using VIVADO-HLS on the following C-code:

```
#define SQM 7
  for(k = 0; k < SQM; k + +){
   #pragma HLS unroll factor = 7
    for(i = 0; i < SQM; i + +){
     #pragm HLS unroll factor = 7
      for(j = 0; j < SQM; j + +){
       #pragma HLS unroll factor = 7
        if (mat[i][j]||||(mat[i][k]&&
           mat[k][j]))  mat[i][j] = 1; }
    }
  }
return(mat[1][SQM − 1]);
```

The synthesys results of the Verilog code obtained for the above code are: $clock - latency = 6ns$, $\#registers = 591$, $\#LUTs = 742$ and $\#MUXes = 782$. This is significantly larger than dnf_y of **Figure 5** which is a DNF with 143 gates and clock latency of less than $1ns$. The experiments show that using a fast pre-computation function of the inputs (COUNTING-1 s) can significantly reduce the size of $\max_y |dnf_y|$ compare to the size of the complete circuit.

The above dnf_y could have been further simplified by replacing monochromatic rectangles with new variables. As illustrated in the **Figure 6**, it is possible to simplify the largest dnf_y of a five node graph by replacing eight monochromatic rectangles (left-side) with new variables. The result is a reduction from $(94/14)/simeq6$ to $(49/14)/simeq3$ in the average number of variables per minterm. As each rectangle is evaluated at runtime, its subset of variables is logically ANDed. This can be achieved with a sub-bus of the FPRM that allows a zero-x_i to broadcast, on the value of 0.

3. Computing the FPRM layout of $\mathrm{dnf_y}$

Using the algorithm of **Figure 2**, we can evaluate the dnf_ys on the FPRM in three steps. Broadcasts representing the minterms of dnf_y are used to evaluate the DNF. As a result, we can evaluate the DNF using the $n \times k$ grid of sub-FPRM, where n is the number of rows (minterms), and k is the number of variables (14×17 for the DNF shown in **Figure 6** right). There are two steps involved: broadcasting the values of each x_i over the columns in the sub-FPRM; and configuring each row as a single bus and computing the logical AND on each row. We can, however, pack several minterms/bus segments in one row, reducing the size of the FPRM sub-grid needed for the computation. Our 2D layout of a dnf_y can be optimized by swapping minterms in each level and arranging the literals in each minterm (node).

Figure 7 illustrates the optimized (by hand) layout of the DNF of **Figure 5** (called LG), wherein the minterms are arranged in six levels, each containing 1–4 minterms. Straight busses are used to broadcast the values of the literals in this layout. According to **Figure 7**, this layout also includes the extra duplications of $'b'$ and $'n'$ required. There is a significant improvement in the total area and max-switching length when compared to the simple method of arranging all minterms in one column. The optimized (by-hand) layout of **Figure 7** is also better compared to that of **Figure 5** when used as an LG. In the following sections, we describe the details of the proposed algorithm to find a minimized LG.

3.1 First stage: find the level arrangement of minterms in the final FPRM layout

1. We build an intersection graph G^0 where each node corresponds to one minterm and each edge corresponds to an intersection between two such minterms. **Figure 8** (left) illustrates this graph for the DNF of **Figure 6**. The edges are labeled by the intersection size.

2. Next, we compute a maximal independent set (MIS) in G^0 that also maximizes the highest label edge in each of its nodes (as depicted in **Figure 8**). This MIS will be used as the first level in the FPRM layout we seek to build. Note that all the minterms in this MIS have no intersection in their variables, thus can be safely mapped to the same level of the layout.

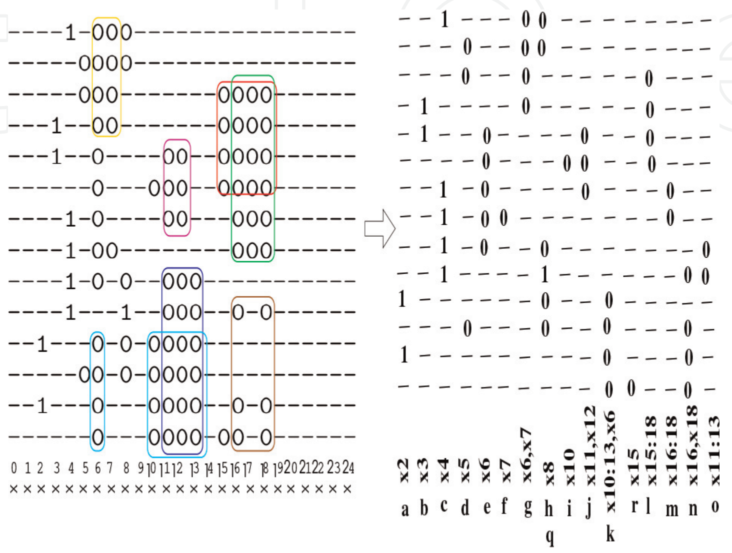


Figure 6.
Reducing the DNF size of a dnf_y by pre-computing monochromatic rectangles.

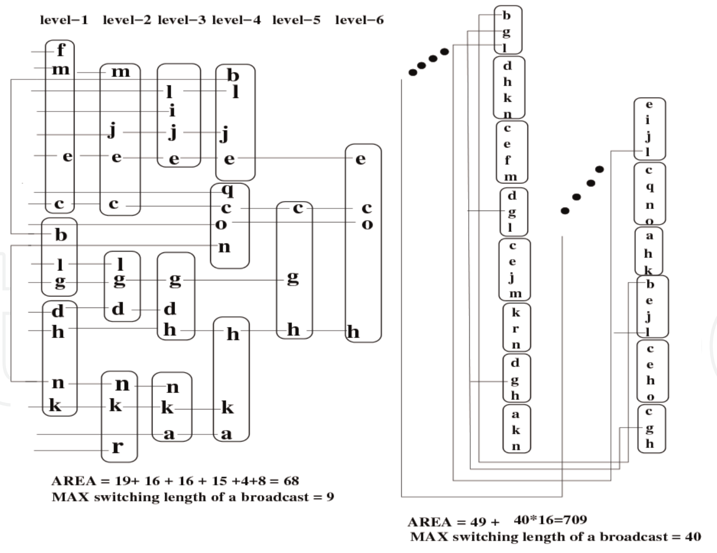

level−1 level−2 level−3 level−4 level−5 level−6
AREA = 19+ 16 + 16 + 15 +4+8 = 68
MAX switching length of a broadcast = 9
AREA = 49 + 40*16=709
MAX switching length of a broadcast = 40

Figure 7.
Optimized FPRM layout.

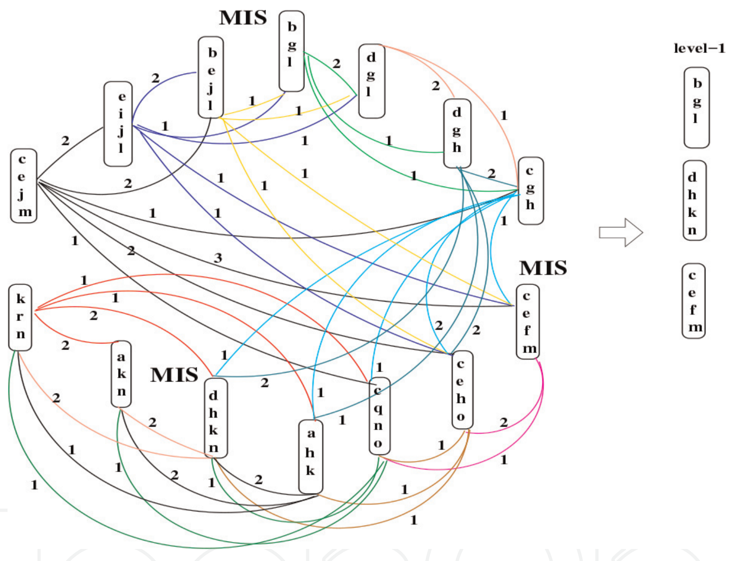


Figure 8.
The intersection graph G° and extracting first MIS.

3. We remove the MIS from G^0 creating G^1 and as depicted in **Figure 9**. compute the next MIS in G^1 creating the next level in the FPRM layout. This process of extracting MIS, forming the next level of minterms in the layout is repeated until there are no more MISs. **Figure 10** depicts the last step of this process.

3.2 Second stage: rearranging the minterms in each level

The position/index of each node/minterm in the final layout is computed as follows:

- Create a leveled graph LG whose nodes $V_{level,index}$ correspond to the minterms in each column of the layout previously obtained.

```
$for(i = last\_level; i >= first\_level; i--)$
 $for(j = first\_index\_in\_level\_i;$
  $j < last\_index\_in_{l}evel\_i; j++)$
  $for(k = i - 1; k > 0; k--)$
   $for(l = first\_index\_in\_level\_k;$
    $l < last\_index\_in\_level\_k; k++)$ {
    add edge $v_{i,j} \longrightarrow v_{k,l}$ if there
     is a subset of literals in $v_{i,j}$
    that is also in $v_{k,l}$ and not covered
    by a previous edge leaving $v_{i,j}$; }
```

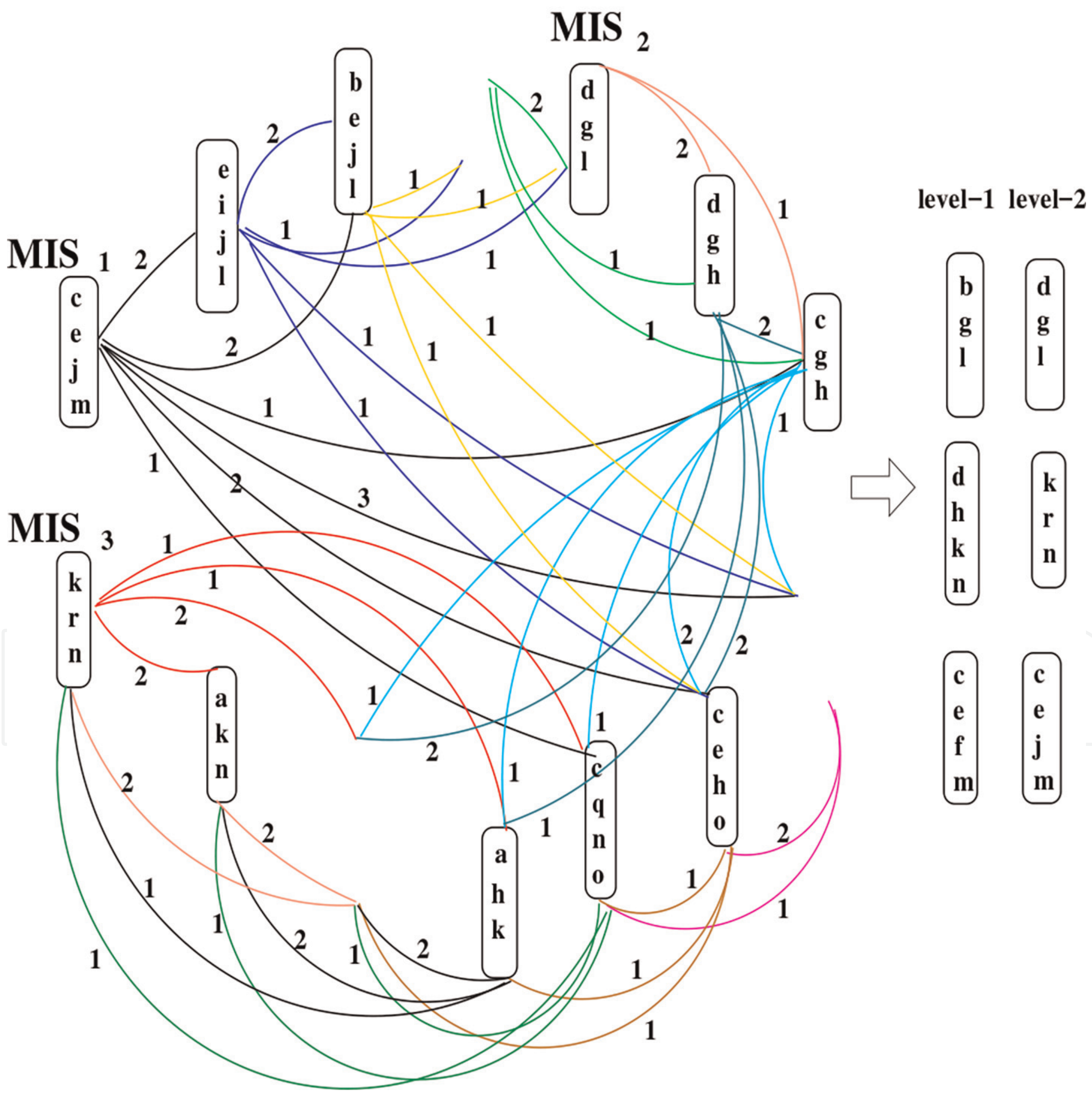


Figure 9.
G^2 and extracting the next level in the layout.

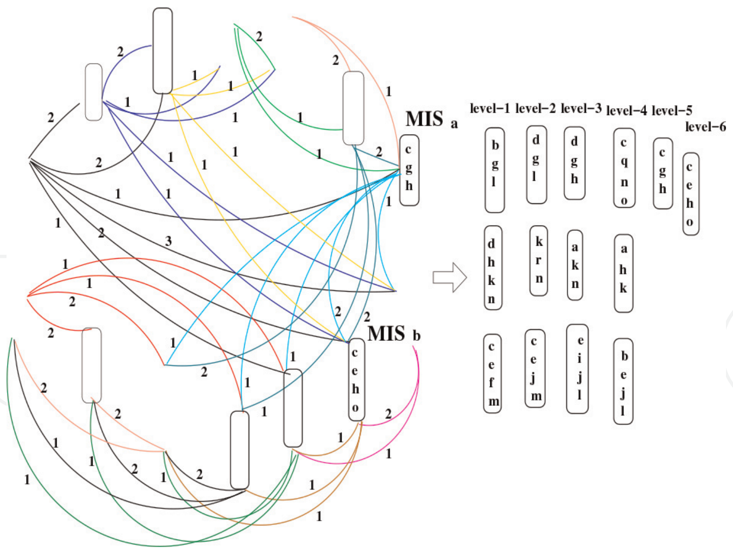


Figure 10.
G^5 and extracting the last level in the layout.

Figure 11 depicts the resulting *LG*.

- Rearrange the minterms in each level of *LG* by: Finding a set of nodes (called "mid-cut"), one from each level of *LG*, and a partition of the remaining nodes in each level into a "left-part" and a "right-part" such that:
    - The number of edges between the left-part and the right-part is minimal.
    - The number of nodes in the left-part in each level is about the same as the number of nodes in the right-part of that level.

Figure 12 depicts finding a mid-cut and the resulting partition to a left-part and right-part. As can be seen, only one edge (the $'b'$) crosses from the left-part to the right-part in **Figure 12**. In the final FPRM layout, the mid-cut minterms will be stacked one on top of the other. Recursively, the process is applied to the left-part and the right-part until all nodes of *LG* are arranged in 2D.

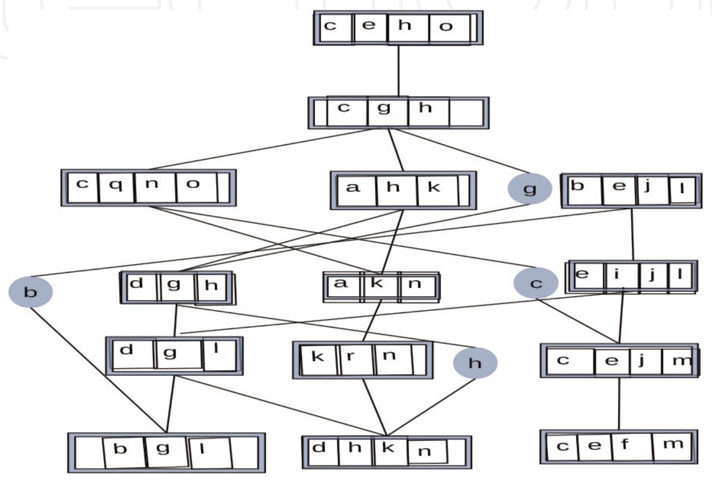


Figure 11.
The leveled graph (LG) of the MIS arrangment.

3.3 Third stage: rearranging the order of literals in each minterm

1. For the current order of literals in each minterm, we expand the edges of the current *LG* to show the duplication of each literal from one level to another. We also add source-edges (arrows in **Figure 13**) depicting that the source of each literal comes from the lowest level below the present 2D layout of the *LG*. **Figure 13** illustrates the resulting graph called the literals graph *TG*.

2. Crossing edges between minterms that are in the same index at their level are eliminated by rearranging the literals inside one minterm.

3. Next we eliminate crossing edges by:

   - Replacing one pair of crossing edges with a down-going source-edge.

   - Reordering literals in the two minterms of the crossing edge. Crossing edges between minterms with the same level-index are resolved by rearranging the literals in those minterms.

The next edge selected to be replaced by a source edge is the one with the maximal number of crossings. For example, in **Figure 13** the first edge to be replaced by a source edge is the edge connecting the left *'b'* in the first level to the right *'b'* in the fourth level, as this edge cut acrosses seven edges. The process is repeated until there are no crossing edges, as shown in **Figure 14**. Since source edges will be aligned vertically later, crossing with source edges is not counted.

3.4 Fourth stage: completing the alignment

At this stage, the minterms in each level and the literals in each minterm have been arranged so that no crossing edges exist. Aligning the literals such that all the edges form straight vertical columns leads to the final FPRM layout:

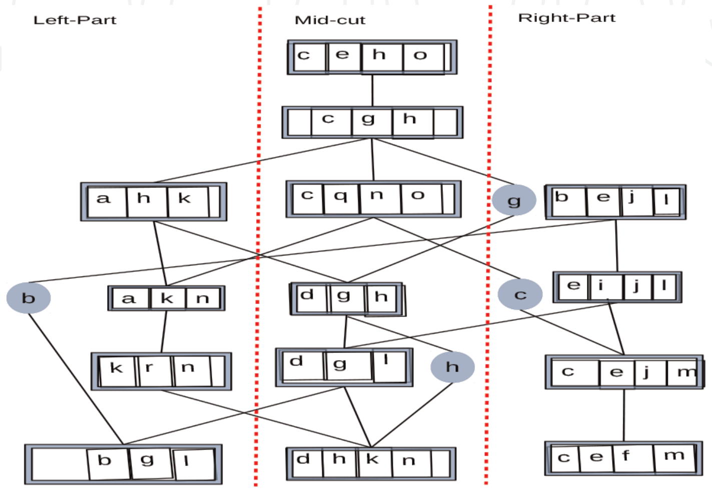


Figure 12.
Rearanging the minterms in LG's levels via mid-cuts.

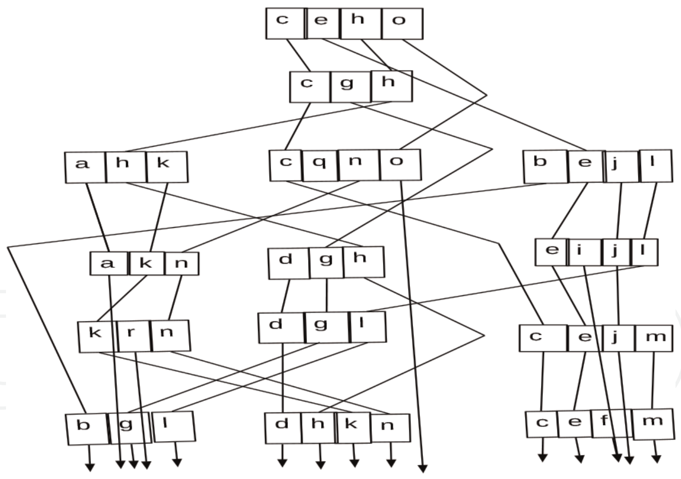


Figure 13.
Expanding current level graph LG to a literal graph TG.

1. We start by placing the bottom leftmost literal/source-edge at the bottom leftmost corner of the FPRM layout. We align all the edges connected to the left corner at a vertical "duplication" column (for duplicating literals, if needed).

2. Let v_i be the nearest node to the last aligned duplication column. We align v_i and the literal/source edges connected to it, into an adjacent vertical duplication column.

3. This is repeated until all edges are aligned into adjacent duplication columns.

Figure 15 illustrates the resulting layout with a total area of about 143, which includes the area for broadcasting the duplicated literals (c,h,b,k,n,l).

Since the FPRMs constructed with literals as control entries into the tristate switches.

4. Realization and results

Tri-state switch is the natural candidate for the infrastructure logic realization of the final FPRM layout, as other switching devices such as NMOS are not supported by

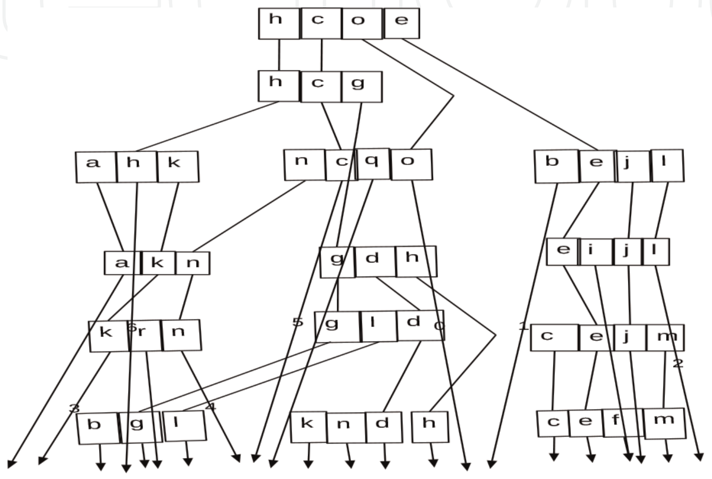


Figure 14.
Final arrangement of literals in each minterm.

ASIC synthesis tools. For control input of '1', the output of the tri-state is exactly identical to its input, but for control input of '0' the output is high impedance (disconnected or 'z').As a result, multiple tri-states can share the same output wires. The DNF is simply a $\vee_{i=1}^{n} m_i$, where m_i represent a minterm $m_i = \wedge_{j=1}^{|m_i|} a_j$, and a_j is a literal. For each literal, a minterm can be represented conceptually by a list of connected tri-states. The leftmost tri-state outputs '1' or 'z' based on input of '1' and control from the literal. The next tri-state, produces the input according to the next literal value in m_i, thus performs the operation of $\wedge_{j=1}^{|m_i|} a_j$. Since the output of each minterm m_i is '1' or 'z', their output wires can be connected directly, performing $\vee_{i=1}^{n} m_i$. **Figure 16** illustrates the FPRM for the DNF of two minterms $M = (a1 \wedge a2) \vee (a2 \wedge a3)$, where the literal values are represented by thick vertical lines (dark-gray for '1' and light-gray for '0'). Each (potential) literal a_j in a minterm m_i residing in row r consists of 6 tri-states and one encoder (depicted in the dashed line rectangle). The $cn_{r,j}$ tri-state is connecting the literal value to minterm m_i, providing

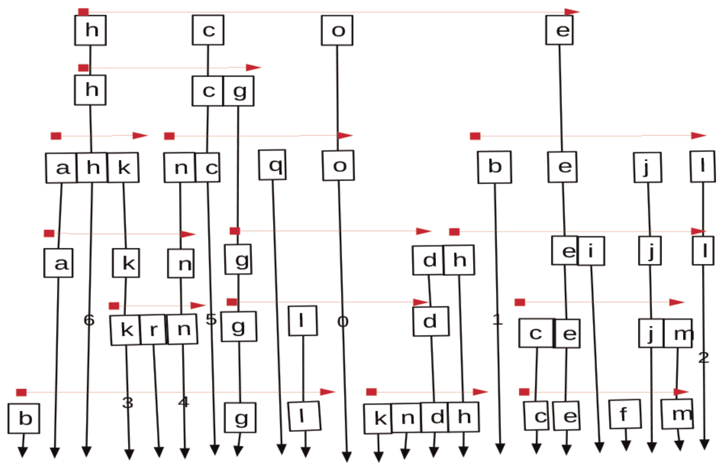


Figure 15.
Completing the alignment.

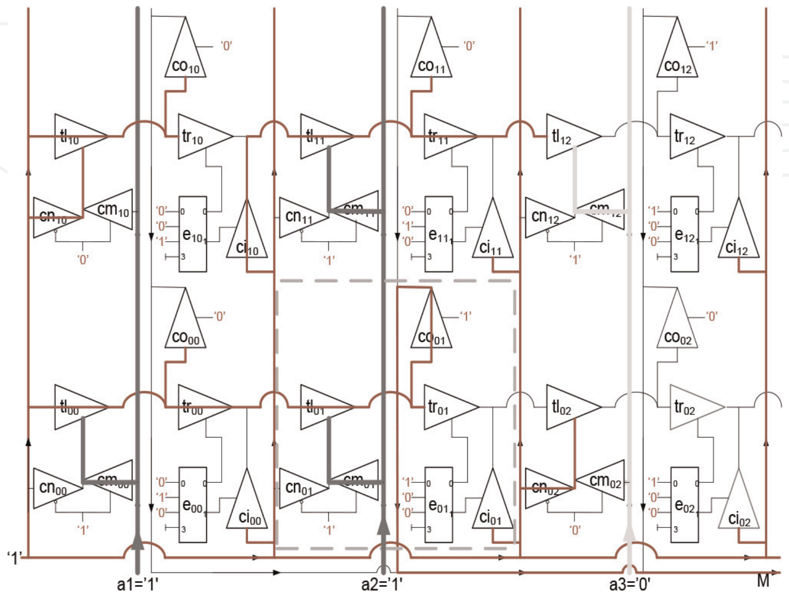


Figure 16.
Tri-state configuration of the FPRM.

that a_j appears in m_i. Otherwise, $cn_{r,j}$ will pass the incoming signal to the next literal. The output of $cn_{r,j}$ is the control of $tl_{r,j}$, transferring the input from a_{j-1} or disconnecting (producing 'z') given the value of a_j. According to the encoder's $e_{r,j}$ input, $tr_{r,j}$ will pass/hold the current signal to a_{j+1}, or $ci_{r,j}$ will start a new minterm calculation, sending '1' to the next literal on the right. Finally, the role of $co_{r,j}$ is to send down the output of the current minterm, providing that it is the rightmost literal in the current minterm. Note that the outputs of the minterms are connected together since these are outputs of tri-states ('1' or 'z'). The output of the DNF is indicated by M on the right-bottom side.

Based on the description in [30], the FPRM implementation is compared to an Island FPGA routing architecture. **Figure 17** shows a variant that contains: A logic unit with and/or-gate that is connected to a grid of $4 - bits\ N\ vertical\ buses \times 4 - bits\ N\ horizontal\ buses$ via two connection units. Any vertical-bus can be connected to any horizontal-bus using a crossbar-like routing unit. Additionally, vertical/horizontal busses can be disconnected, so that bends will not consume the entire bus.

All connections/disconnections and fuse operations are made by a back-to-back pair of tri-state devices, allowing bi-directional signals. ASIC synthesis results obtained with Synopsys Design compiler using a 160 nm cell library. As shown in **Table 1**, the FPRM architecture is 4X faster and more efficient in both power and area[3] than the FPGA routing infrastructure. Based on the FPRM of **Figure 16** and

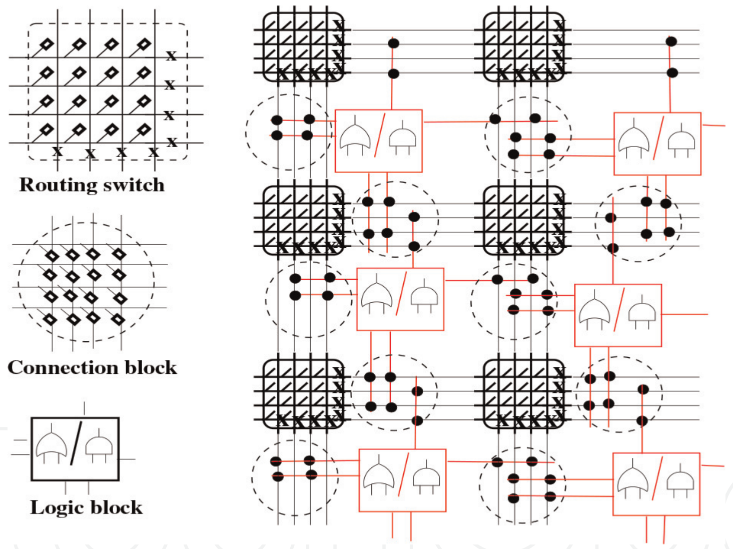


Figure 17.
The FPGA routing architecture used for the experiments.

Size	Area		Power		Clock latency	
	FPRM	FPGA	FPRM	FPGA	FPRM	FPGA
16 × 16	6156 uC	38,256 uC	66 uW	161 uW	6.98 ns	14.36 ns
12 × 12	3463 uC	21,528 uC	52 uW	149 uW	4.75 ns	11.27 ns
8 × 8	1221 uC	9579 uC	34 uW	148 uW	2.85 ns	8.18 ns
4 × 4	333 uC	2411 uC	12 uW	80 uW	1.36 ns	5.09 ns

Table 1.
Synthesis results comparing the FPRM vs. the FPGA routing infrastructure.

the assumption that the counting stage requires two cycles. When the expected latency of the FPGA is added, we get about twice as fast performance from the FPRM.

A chain of switches (tri-states) selects whether values should be passed on or not in the circuit we designed. This idea will obviously work faster than a chain of *and* and *or* gates as implemented in FPGA. The fact that there are no switches along the wire that ends with M further accelerates the speed of receiving the output. This is triggered when a value of 1 comes out from one of the minterms. Given that the tri-state consumes power as an ordinary buffer, and the *and*/*or* operations ($\vee_{i=1}^{n} mintermi$) are implemented simply by merging the tri-states outputs, the power consumption is likely to be a function of the number of tri-state buffers. On the other hand, the FPGA needs to be powered for the *and*/*or* gates as well as the switching systems to connect the logical blocks. Compared to a real FPGA, we have simplified our implementation, but this can only reduce power. Conversely, the FPRM is general, assuming that any Boolean Circuit can be represented as a DNF.

5. Conclusions

As part of the contribution of this work, we developed the algorithm to evaluate boolean circuits on the RM; a method to compute an optimized FPRM layout; and a method for realizing the FPRM as a tri-state circuit with comparable performance to the conventional FPGA implementation. A tri-state (MOSFET transistor) acts as a switching element in both the FPGA and FPRM. Passing a signal through a chain of k switches (that is, a chain of k source-drain connected transistors) incurs a quadratic delay of $\frac{k^2}{2} r \cdot c$ (where r is the resistance and c is the capacitance of each transistor). As a result of the reconfiguration of the FPRM, a relatively long chain of transistors can be created. Due to the short chains involved in the circuit evaluation problem discussed here, the FPRM will be able to execute the circuit evaluation process fairly quickly. In order to compare the FPRM with the FPGA/ASIC realization of $f(x_1, \dots, x_n)$, a SPICE simulation of the FPRM can be carried out. This includes selecting the most appropriate MOSFET transistor technology to minimize signal propagation delays through the FPRM bus.

This research can be furthered by comparing the synthesized results with those obtained from HLS (High Level Synthesis) of the $f(x_0, \dots, x_{n-1})$ C-code. Analyzing other functions that can be efficiently computed using the FPRM in $O(1)$. Finally, study how the partitioning into segments affects the size of the resulting formulas, and build a decision tree that computes $f(x_0, \dots, x_{n-1})$ on the FPRM in an even smaller size.

Appendix

A. List of acronyms and abbreviations

- **ASIC:** Application Specific Integrated Circuit
- **BC:** Boolean Circuit
- **CLU:** Configurable Logic Unit

- **CMOS:** Complementary Metal-Oxide Semiconductor
- **DNF:** Disjunction Normal Form
- **DR:** Dynamic Reconfiguration
- **DRFPGA:** Dynamic Reconfiguration FPGA
- **FDR:** Fine-Grain Dynamically Reconfigurable Architecture
- **FPRM:** Field Progammable Reconfigurable Mesh
- **HLS:** High Level Synthesis
- **LE:** Logic Elements
- **LR-Mesh:** Linear Reconfigurable Mesh
- **LUT:** Lookup Table
- **MIS:** Maximal Independence Set
- **MOSFET:** Metal-Oxide Semiconductor Field-Effect Transistor
- **NMOS:** N-type Metal-Oxide Semiconductor
- **PE:** Processing Elements
- **PR:** Partial Reconfiguration
- **PRAM:** Parallel Random Access Machine
- **RM:** Reconfigurable Mesh
- **RMBM:** Reconfigurable Multiple Bus Machine
- **RRAM:** Resistive Random Access Memory
- **SRGA:** Self Reconfigurable Gate Array
- **STCON:** st-connectivity

Author details

Esti Stein[1*†] and Yosi Ben Asher[2†]

1 Department of Computer Science, The Academic College of Tel Aviv-Yaffo, Jaffa, Israel

2 Department of Computer Science, Haifa University, Haifa, Israel

*Address all correspondence to: esterst@mta.ac.il

† These authors contributed equally.

References

[1] Asher Y, B, Stein E. Evaluation of circuits on the reconfigurable mesh. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Rio De Janeiro, Brazil: IEEE; 2019. pp. 71-74

[2] Vaidyanathan R, Trahan J. Dynamic Reconfiguration: Architectures and Algorithms. US: Springer Science & Business Media; 2004

[3] Matias Y and Schuster A. On the Power of a 2-Band Reconfigurable Network. Unpublished Manuscript. 1992

[4] Chen G, Wang B, Li H. Deriving algorithms on reconfigurable networks based on function decomposition. Theoretical Computer Science. 1993; **120**(2):215-227

[5] Nakano K, Wada K. Integer summing algorithms on reconfigurable meshes. Theoretical Computer Science. 1998;**197**: 57-77

[6] Jang J, Park H, Prasanna VK. An optimal multiplication algorithm on reconfigurable mesh. In: Proc. Symp. On Parallel and Distributed Processing. Beverly Hills, CA: IEEE; 1992. pp. 381-391

[7] Jang J, Prasanna VK. An optimal sorting algorithm on reconfigurable mesh. In: Proc. Inter. Parallel Processing Symp. Beverly Hills, CA: IEEE; 1992. pp. 130-137

[8] Elmesbahi M, KJ, Errami A, Bouattane O. Theta(1) time parallel algorithm for finding 2d convex hull on a reconfigurable mesh computer architecture. Global Journal of Computer Science and Technology. 2021;**21**:1-9

[9] Trahan JL, Subbaraman CP, Vaidyanathan R. List ranking and graph algorithms on the reconfigurable multiple machine. In: Proceedings of International Conference on Parallel Processing. NY: Syracuse University, CRC Press; 1993. pp. III–224-III–247

[10] Wang B-F, Chen G-H. Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. IEEE Transactions on Parallel and Distributed Systems. 1990;**1**(4): 500-507

[11] Miller R, Prasanna-Kumar VK, Reisis DI, Stout QF. Image computations on reconfigurable VLSI arrays. In: Proceedings of the Conference on Vision and Pattern Recognition. Ann Arbor, MI: IEEE; 1988. pp. 925-930

[12] Trahan JL, Vaidyanathan R. Relative scalability of the reconfigurable multiple bus machine. In: Proc. Workshop Reconfigurable Arch. And Algs. Honolulu, HI: IEEE; 1996

[13] Sidhu R, Wadhwa S, Mei A, Prasanna VK. A self-reconfigurable gate array architecture. In: Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing. Berlin, Heidelberg: Springer; 2000. pp. 106-120

[14] Hatem ME-B, Vaidyanathan R, Trahan JL, Rai S. On the communication capability of the self-reconfigurable gate array architecture. IPDPS. 2002:**500**: 0152b. IEEE

[15] Hatem ME-B, Vaidyanathan R, Trahan JL, Rai S. On designing implementable algorithms for the linear reconfigurable mesh. PDPTA. 2003: 241-246

[16] Ben-Asher Y, Stein E, Tartakovsky V. Fpga realization of the reconfigurable mesh counting algorithm. Journal of Circuits, Systems and Computers. 2021;**30**(9):2150157

[17] Giefers H, Platzner M. Armlang: A language and compiler for programming reconfigurable mesh many-cores. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium. Rome, Italy: IEEE; 2009. pp. 1-8

[18] Ben-Asher Y, Stein E, Vaidyanathan R. Combining boolean gates and branching programs in one model can lead to faster circuits. In: Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International. Orlando, FL: IEEE; 2017. pp. 184-191

[19] Giefers H, Platzner M. An Fpga-Based Reconfigurable Mesh Many-Core. IEEE Transactions on Computers. 2013; **63**(12):2919-2932

[20] Hauck S, Fry TW, Hosler MM, Kao JP. The chimaera reconfigurable functional unit. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2004;**12**(2):206-217

[21] Xilinx. Logicore Ip Xps Hwicap. Report DS586. San Jose, CA: Xilinx; 2010

[22] Intel Corporation. Intel® Quartus® Prime Pro Edition User Guide: Partial Reconfiguration. San Jose, CA: Intel; 2022

[23] Babu P, Parthasarathy E. Reconfigurable fpga architectures: A survey and applications. Journal of The Institution of Engineers (India): Series B. 2021;**102**(1):143-156

[24] Khan MA, Miyamoto N, Pantonial R, Kotani K, Sugawa S, Ohmi T. Improving multi-context execution speed on drfpgas. In: Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian. San Francisco, CA: IEEE; 2006. pp. 275-278

[25] Cong J, Xiao B. A novel fpga architecture with memristor-based reconfiguration. In: Nanoscale Architectures (NANOARCH), 2011 IEEE/ACM International Symposium. San Diego, CA: IEEE; 2011. pp. 1-8

[26] Cong J, Xiao B. Fpga-rpi: A novel fpga architecture with rram-based programmable interconnects. IEEE Trans. VLSI Syst. 2014;**22**(4):864-877

[27] Lin T-J, Zhang W, Jha NK. A fine-grain dynamically reconfigurable architecture aimed at reducing the fpga-asic gaps. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2014;**22**(12):2607-2620

[28] Ben-Asher Y, Stein E. Adaptive booth algorithm for three-integers multiplication for reconfigurable mesh. Journal of Interconnection Networks. 2016;**16**(1):1-25

[29] Rudell R, Sangiovanni-Vincentelli A. Espresso-mv: Algorithms for multiple-valued logic minimization. Proc. IEEE Custom Integrated Circuits Conf. 1985: 230-234

[30] Xilinx. The Programmable Logic Data Book. 2000. Available from: http://www.xilinx.com/index.shtml.